

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平8-263262

(43) 公開日 平成8年(1996)10月11日

(51) Int.Cl. <sup>6</sup>	識別記号	庁内整理番号	F I	技術表示箇所
G 0 6 F 5/00			G 0 6 F 5/00	H
G 0 6 T 9/00		9382-5K	H 0 3 M 7/46	
H 0 3 M 7/46			H 0 4 N 1/413	Z
H 0 4 N 1/413			G 0 6 F 15/66	3 3 0 E

審査請求 未請求 請求項の数 2 O L (全 10 頁)

(21) 出願番号 特願平7-60900

(22) 出願日 平成7年(1995)3月20日

(71) 出願人 591044164

株式会社沖データ

東京都港区芝浦四丁目11番地22号

(72) 発明者 ▲高▼橋 芳典

東京都港区芝浦4丁目11番地22号 株式会

社沖データ内

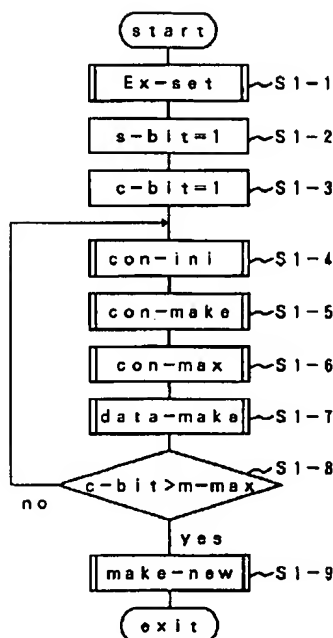
(74) 代理人 弁理士 鈴木 敏明

(54) 【発明の名称】 印刷データの圧縮方法

(57) 【要約】

【目的】 変換を行う行のデータとその直前行のデータとがほとんど一致しない場合でも、効率的に圧縮が行える圧縮方法を提供する。

【構成】 まず変換を行う当該行とn行前の同じビット位置のデータの排他的論理和Ex(n,m)を求める(ステップ1-1)。そして未変換の先頭ビット位置変数s-bitおよび検索中ビット位置を示すポインタの変数c-bitの初期化を行って(ステップ1-2、1-3)、連続同様ビット数con(n)を初期化する(ステップ1-4)。次にサブルーチンcon-makeをコールして、未変換先頭ビット位置からの先行各行での連続同様ビット数を検索してその結果をcon(n)に格納する(ステップ1-5)。生成されたcon(n)のうち最大のものを探し(ステップ1-6)、結果として得られた最大連続同様ビット数にしたがって、サブルーチンdata-makeをコールして変換データを生成する(ステップ1-7)。



実施例の動作を示すフローチャート

## 【特許請求の範囲】

【請求項1】 印刷データを圧縮する方法において、変換を行う当該行の印刷データと同様のデータが当該行に先行する複数の行にあるか否かを検索し、前記先行する複数の行の各々において連続する同様のデータの数を算出し、

前記先行する複数の行のうち連続する同様のデータの数の最も多い最長連続同様データ保持行と前記当該行の連続する同様のデータ数が所定数以上同様である場合、前記保持行の同様データで前記当該行のデータを変換すること

10

を特徴とする印刷データの圧縮方法。

【請求項2】 前記最長連続同様データ保持行の同様データでの前記当該行のデータの変換は、該保持行の同様データで変換することを示すビットと、連続同様ビット数を示すビットと、該保持行の先行行数を示すビットとにより行う請求項1記載の印刷データの圧縮方法。

## 【発明の詳細な説明】

## 【0001】

【産業上の利用分野】 本発明は、プリンタで印刷を行う印刷データを圧縮する方法に関し、とくに印刷データを保管する際に圧縮する方法に関する。

20

## 【0002】

【従来の技術】 図2は印刷結果を示す説明図、図3は図2の印刷結果を得るための印刷データを示す説明図である。図2は印刷可能な最小物理サイズ素点（ドット）の集まりを表現したもので、図3はドットを2進値（1または0の情報、ビット）の集まりとして表現している。

【0003】 図3において、左から12番目までの1、0の組み合わせは、第6行以降変化していない。そこで各々の行のデータを直前の行のデータと比較し、直前の行と同じビットの組み合わせの部分は、「ここからの連続するmビットの組み合わせは、直前行と同じである」という情報に変換し、直前の行とビットの組み合わせが異なる部分は、「ここからの連続するnビットは、続くnビットの情報を充てる」という情報として格納することにより、変換する前よりも少ないビット数で再生可能なビット列が生成できる。この様な圧縮方法では、続くデータが直前行と同じ情報を示すのか否か、あるいは何ビット分が直前行と同じなのかを示すためのビットが必要になる。

30

## 【0004】

【発明が解決しようとする課題】 しかしながら上記従来の圧縮方法では、図4に示すようなドットに印刷する場合はデータを圧縮できない。図5は図4のドットに対応する2進値表である。図5からわかるように、各々の行のデータは直前行のデータとほとんど一致しない。したがって、上記従来の方法で圧縮すると、続くデータが直前行と同じ情報を示すのか否か、あるいは何ビット分が直前行と同じなのかを示すためのビットが必要になる分、ビット数が増加することになる。

40

## 【0005】

【課題を解決するための手段】 上記課題を解決するために本発明の圧縮方法は、変換を行う当該行の印刷データと同様のデータが当該行に先行する複数の行にあるか否かを検索し、前記先行する複数の行の各々において連続する同様のデータの数を算出し、前記先行する複数の行のうち連続する同様のデータの数の最も多い最長連続同様データ保持行と前記当該行の連続する同様のデータ数が所定数以上同様である場合、前記保持行の同様データで前記当該行のデータを変換するようにしたものである。

## 【0006】

【作用】 上記構成を有する本発明によれば、変換を行う当該行の印刷データと同様のデータが当該行に先行する複数の行にあるか否かを比較し、その中から、当該行の印刷データと同様の連続するデータを有する先行行のデータを指定して圧縮するので、効率的に圧縮を行うことができる。

## 【0007】

【実施例】 以下、本発明に係る実施例を図面にしたがって説明する。なお各図面に共通する要素には同一の符号を付す。図1は本発明に係る実施例の動作を示すフローチャートである。

【0008】 まず図5において、第3行のドット組み合わせは、直前の第2行と比較すると、第1列、2列、5～29の奇数列、30列および32列が同じであるが、第1行と比較すると、第4～第32列までのビット列が全く同じである。この様に、該当行に先行する複数行まで比較の範囲を広げることにより、「続くnビットはm行前の行の同じビット位置のデータと同じである」という情報に変換し、前の複数行まで比較してもなお該当行と同じビットの組み合わせが存在しない場合は、「ここからの連続するnビットは続くnビットの情報を充てる」という情報として格納することにより、変換する前よりも少ないビット数で再生可能なビット列が生成できる。

【0009】 上記2種類の情報を混在させるために本実施例では図6、図7に示す様式を使用する。図6は様式①を示し、図7は様式②を示す。本実施例では、1つの様式で指定する最大ビット数は16、比較の対象とする先行行数は最大8としている。これら2つの数値は、変換の対象となる実際のドット列の組み合わせにより最も効率の良い値が異なる。

【0010】 本実施例では、続くデータが先行行と同じ情報を示すのか否かを表すために1ビットを充てる。図6に示す様式①601において、“1”は先行行と同じ情報であることを示し、“0”は新たなビットの組み合わせであることを示す。例えば図6においては、様式識別ビット602は“1”となっているので、続くデータは先行行と同じ情報である。連続同様ビット数603は、格納される数値+1で何ビット分が連続して先行行

50

と同じ組み合わせを示す。また先行行数604は、格納される数値+1で何行前と連続して同様なのかを示す。したがって、図に示すように、連続同様ビット数603が5で、先行行数604が3の場合は、6ビットデータが4行前の行と連続して同じ情報であることを示している。

【0011】図7において、様式②701では、様式識別ビット702は“0”である。挿入ビット数703は、格納される数値+1で何ビット分が連続して挿入されるかを示す。挿入データ704は、挿入されるべきデータそのものを示す。即ち図7に示すように、挿入ビット数703が4である場合、5ビットデータ704が連続して新たに挿入されることを示している。

【0012】図5に示す2進値表の第1行目のデータは、比較する対象がないので、すべて様式②となり、したがって様式識別ビット702は“0”、挿入ビット数703は15、挿入データ704は2進値で“0111010101010101”、続いて同じ様式②で挿入データ704は、“010101010101011”となる。この結果、第1行のデータは図8に示す通りとなる。

【0013】第2行目については、第1行と比較すると同様ビットが分散した4ビットのみなので、第1行と同じくすべて様式②となり、第2行データは図9に示す通りの結果となる。

【0014】また第3行については、最初の2ビットは第2行と同様であるが、様式①で表現すると8ビットを要し、様式②の場合の7ビットよりもビット数が増加する。そこで第3ビット目を含めて様式②で表現する。第4ビット～第32ビットはすべて第1行と同様なので様式①で表現する。第4ビット～第19ビットは「連続する16ビットは2行前と同様」という内容となるので、様式識別ビットは602は“1”、同様ビット数は15、先行行数は1となる。第20ビット～第32ビットは「連続する13ビットは2行前と同様」という内容となるので、様式識別ビットは602は“1”、同様ビット数は12、先行行数は1となる。この結果、第3行データは図10に示す通りとなる。

【0015】ここで何ビット連続して同様の場合に様式②よりも様式①を採用した方が必要ビット数が少ないかを考えてみる。様式①は固定的に8ビットを必要とする。連続するxビットを様式②で表現すると、xビットのデータそのものに5ビットが付加されるので、計(x+5)ビットが必要となる。このビット数が8以上、即ちxが4以上、つまり4ビット以上が連続して先行行と同様の場合に様式①を採用すると、必要ビット数が様式②と同じかまたは様式②より少なくなる。

【0016】図11は本発明の実施例を適用するプリンタのブロック図である。同図において、CPU111はプリンタに動作全体を制御するとともに、本実施例における印刷データの圧縮を行う。CPU111にはバス1

12を介してROM113、RAM114およびポート115が接続されている。ROM113には制御プログラム等が格納され、RAM114には印刷データが一時的に格納される。ポート115には、上位インタフェース回路116および印刷機構部駆動回路117が接続され、印刷機構部駆動回路117にはさらに印刷機構部118が接続されている。上位インタフェース回路116には図示しない上位装置が接続されており、この上位装置からプリンタに印刷データが送られてくる。

【0017】次に各フローチャートにしたがって印刷データの圧縮動作を説明する。図1に示すフローチャートは、本実施例のアルゴリズム全体を示すメインルーチンであり、いくつかのステップにおいてサブルーチン呼び出す。ここで、各サブルーチンで使用されるアルゴリズムについて説明する。

【0018】まず、 $cur(m)$ は当該行の第mビットを表し、 $Bit(n, m)$ は当該行のn行前の第mビットを表す。つまりここでは、 $cur(m)$ と $Bit(n, m)$ には予め相当するビットデータが格納されているという前提にたっている。 $Ex(n, m)$ は当該行とn行前の同じビット位置のデータの排他的論理和(記号#は排他的論理和の演算子である)であり、 $Ex(n, m)$ が0の場合、当該行の第mビットと同じデータがn行前にあるといえる。 $con(n)$ はn行前に検索中のビット位置から連続して存在する当該行と同じビットの組み合わせ数が格納される。 $s-bit$ は未変換の先頭ビット位置の変換であり、 $c-bit$ は検索中ビット位置を示すポインタの変数である。 $m-max$ は当該行の総ビット幅を表す。なお使用する様式は、図6および図7に示す様式①と様式②としているので、同様ビットを検索する先行行数604は8で、連続同様ビット数603および挿入ビット数703の最大値は16としている。また、当該行に先行する行の数が8未満の場合の例外処理は以下に説明する動作には含まれていない。

【0019】まず図1のフローチャートにおいて、サブルーチン $Ex-set$ を呼び出して $Ex(n, m)$ を生成する。図12はサブルーチン $Ex-set$ を示すフローチャートである。図12において、まず先行行のナンバーを表す変数iを1に初期化し(ステップ12-1)、次にビット位置を表す変数jを1に初期化する(ステップ12-2)。ステップ12-3では、当該行と1行前の最初のビット位置のデータの排他的論理和 $Ex(i, j)$ を求め、ステップ12-4でビット位置を変え、最後のビット位置のデータまで(ステップ12-5)、当該行と1行前の各ビット位置のデータの排他的論理和 $Ex(i, j)$ を求める。当該行と1行前の各ビット位置のデータの排他的論理和 $Ex(i, j)$ が求められると、次に当該行と2行前の各ビット位置のデータの排他的論理和 $Ex(i, j)$ を求める(ステップ12-6)。以上の処理を8行前の各ビット位置のデータについて行う(ステップ12-7)。

【0020】図1において、サブルーチン $Ex-set$ が終了

5

すると、未変換先頭ビット位置変数s-bit を初期化する（ステップ1-2）とともに、検索中ビット位置を示すポインタ変数c-bit の初期化を行う（ステップ1-3）。次にサブルーチンcon-iniを呼び出す（ステップ1-4）。

【0021】図13はサブルーチンcon-ini を示すフローチャートである。同図において、まず先行行のナンバーの変数iを1に初期化し（ステップ13-1）、連続同様ビット数con(i)を初期化して0とする（ステップ13-2）。以上の処理を先行する8行分行う（ステップ13-3、13-4）。これにより連続同様ビット数con(i)としてすべて0は格納される。この処理が終了すると、次にサブルーチンcon-makeをコールする（ステップ1-5）。

【0022】図14はサブルーチンcon-makeを示すフローチャートである。同図において、まず本サブルーチン内で使用するビット位置変数jを未変換先頭ビット位置変数s-bit で初期化する（ステップ14-1）。これにより本サブルーチン内では、s-bit 位置より同様ビットの有無等が検索される。次に先行行のナンバーiを1で初期化する（ステップ14-2）。次にi番目の先行行の検索中ビット位置データが当該行の検索中ビットのデータと同じか否かを判断し（ステップ14-3）、同じならばステップ14-4に分岐し、同じでなければステップ14-11に分岐する。

【0023】ステップ14-4では、同様ビットが1つ見付かったので、検索中の連続同様ビット数con(i)を1とする。その後第1先行行において何ビット連続して同様かを検索し、前記con(i)に格納する。ここでは連続同様ビット数検索中のビット位置を示す変数としてkを用いている。即ち、検索中ビット位置は変数jに在るので、変数kはステップ14-5でjの値に初期化され、次にビット位置を調べるためにステップ14-6で1増加させ、ステップ14-9で実際に当該行と同じデータか否かを調べる。ステップ14-9で同じと判断された場合は、ステップ14-10で連続同様ビット数con(i)を1増加させ、さらに次のビット位置の検索のためにステップ14-6に無条件分岐する。ステップ14-9で当該行と同じでないと判断された場合は、即座に検索ループを抜け、ステップ14-11に分岐して次の先行行での処理に移行する。

【0024】ステップ14-7では、検索中に連続同様ビット数がその最大値16を越えた場合はループを抜けることを示す。また変数kはビット位置を表す変数なので、検索時のその値は最大ビット幅m-max を越える必要がなく、ステップ14-8はビット位置変数kが最大ビット幅m-max を越えるとループを抜けることを示している。以上により、未変換先頭ビット位置からの先行各行での連続同様ビット数の検索結果がcon(n)に格納される。サブルーチンcon-makeの処理が終了すると、つぎに

6

サブルーチンcon-max がコールされる（ステップ1-6）。

【0025】図15はサブルーチンcon-makeを示すフローチャートである。このサブルーチンは、各先行行の連続同様ビット数con(i)のうち最大のものを探す処理を行う。同図において、iは検索中の先行行のナンバーを示し、pは第1先行行検索時点までの最大連続同様ビット数を有する行の先行行ナンバーを保持する変数である。まず先行行ナンバーの変数iを当該行の直前行を表す1に初期化し（ステップ15-1）、最長連続同様ビット保持行ナンバーpを0とする（ステップ15-2）。これにより検索開始時点でいずれの行にも同様ビットが存在しないように設定する。次にステップ15-3で、検索中の行の連続同様ビット数が0か否かを判断し、0の場合はステップ15-7に分岐して次の先行行に検索の対象を移す。

【0026】検索中の行の連続同様ビット数が0でない場合は、既知の最長連続同様ビット数との比較を行うべくステップ15-4に分岐する。ステップ15-4では、既知の最長連続同様ビット数が在るか否かを判断し（最長連続同様ビット保持行ナンバーpが0の場合は、既知の最長連続同様ビット数は存在しない）、既知の最長連続同様ビット数が存在しない場合、ステップ15-6に分岐して検索中の行を最長連続同様ビット保持行ナンバーとするために、pに検索中の行ナンバーiを代入する。

【0027】また既知の最長連続同様ビット数が存在する場合は、ステップ15-5で、既知の最長連続同様ビット数con(p)と検索中の行の連続同様ビット数con(i)とを比較し、検索中の行の連続同様ビット数con(i)が既知の最長連続同様ビット数con(p)よりも大きい場合に、ステップ15-6にて、検索中の行を最長連続同様ビット保持行ナンバーとするために、pに検索中の行のナンバーiを代入する。検索中の行の連続同様ビット数con(i)が既知の最長連続同様ビット数con(p)よりも大きくない場合は、最長連続同様ビット保持行ナンバーpは更新せず、ステップ15-7に分岐して次の先行行に検索の対象を移す。ステップ15-8は、検索対象の先行行の検索を終了したか否かを、検索中の行のナンバーiが8を越えたか否かで判断し、検索対象行数以下の場合は次の先行行の検索のためにステップ15-3に分岐し、検索対象行数8を越えた場合はこのサブルーチンを終了する。本サブルーチンの終了時、変数pには最長連続同様ビット保持行のナンバーが格納される。同様ビットをいずれの検索対象先行行にも有しない場合、変数pは0となっている。サブルーチンcon-max の処理が終了すると、つぎにサブルーチンdata-make がコールされる（ステップ1-7）。

【0028】図16はサブルーチンdata-make を示すフローチャートである。このサブルーチンdata-make は、

サブルーチンcon-max で得られた最長連続同様ビット数にしたがって、変換データを生成する処理を行う。このサブルーチンdata-make の説明に先立って、このサブルーチンdata-make の中でコールされる別のサブルーチンmake-same およびサブルーチンmake-newについて説明する。図17はサブルーチンmake-same を示すフローチャートである。

【0029】図17において、サブルーチンmake-same は、様式①にて先行行に連続同様データが存在する場合の変換データを作成するサブルーチンである。なおこのサブルーチンにおいて、平行四辺形の処理ボックスは、変換データをメモリ（図11に示すRAM114）内の領域に格納する処理を示す。まずステップ17-1で、メモリ内に様式識別ビット602として“1”を格納する。これにより様式①であることを明示する。次にステップ17-2で、検索中ビット位置を示すポインタ変数c-bit と未変換先頭ビット位置変数s-bit の差を連続同様ビット数603としてメモリに格納する。続くステップ17-3では、様式①の先行行数604を格納するが、変数pは直前行のナンバーを1としているので、様式①に  
20 対応させるためにp-1を格納する。

【0030】図18はサブルーチンmake-newを示すフローチャートである。このサブルーチンmake-newは、いずれの検索対象先行行にも同様データが存在しない場合に、様式②で変換データを作成するサブルーチンである。このサブルーチンにおいても、平行四辺形の処理ボックスは、変換データをメモリ（図11に示すRAM114）内の領域に格納する処理を示す。まずステップ18-1で、メモリに様式識別ビット702として“0”を格納し、様式②であることを明示する。ステップ18-2では、検索中ビット位置を示すポインタ変数c-bit と未変換先頭ビット位置変数s-bit の差を挿入ビット数703としてメモリに格納する。検索中ビット位置を示すポインタ変数c-bit を利用して、様式②の挿入ビット数703として (c-bit)-(s-bit) を充てることができる。  
30

【0031】ステップ18-3からステップ18-6では、様式②の挿入データ704を格納する。変数iは格納すべきビット位置のポインタとして使用されており、ステップ18-3で未変換ビット位置変数s-bit を代入して初期化され、ステップ18-4で当該行のビットデータcur(i)を格納し、ステップ18-5で1ずつ増加されるポインタに従って、ステップ18-6でポインタがc-bit を越えるまで連続してメモリに格納する。

【0032】次に図16のフローチャートにしたがってサブルーチンdata-make について説明する。このサブルーチンでは、連続同様ビット数con(p)が4を越える場合に、様式①で変換データを格納して未変換先頭ビット位置変数s-bit を更新し、それ以外の場合は、未変換ビット数((c-bit)-(s-bit+1))が様式②で規定される最大  
40

ビット長16を越えようとする場合に様式②で変換データを格納して未変換先頭ビット位置変数s-bit を更新し、越えない場合は単にc-bit を1増加している。

【0033】まずステップ16-1では、検索対象先行行に同様ビットが存在するか否かを、変数pが0（存在しない場合）か1（存在する場合）かで判定し、存在する場合はステップ16-2に分岐し、存在しない場合はステップ16-3に分岐する。ステップ16-2では、前述したように連続同様ビット数が4以下の場合は様式②の方が変換後のビット数が少なくなるので、連続同様ビット数が4以下の場合はステップ16-3に分岐して同様ビットが存在しない場合と同様の処理を行う。

【0034】ステップ16-2で連続同様ビット数が4を越える場合は、ステップ16-4に分岐し、未変換先頭ビット位置と検索中ビット位置が等しいか否かを判断する。ここで等しくないと判断された場合は、未変換先頭ビット位置と検索中ビットの1ビット前まではデータは様式②に従って変換されなければならないので、ステップ16-5からステップ16-8にて様式②に従って変換される。ステップ16-5では、検索中ビット位置の1ビット前までのデータを様式②に従って変換するために、検索中ビット位置ポインタ変数を1減らす。これにより次にステップ16-6でサブルーチンmake-newを呼び出す準備をする。ステップ16-6ではサブルーチンmake-newを呼び出し、図18で説明したサブルーチンmake-new処理を行い、様式②に従ってデータを変換する。ステップ16-7では、ステップ16-5で補正したc-bit の内容を復旧し、ステップ16-8で、復旧したc-bit の値をs-bit に代入して未変換先頭ビット位置を補正する。  
30

【0035】ステップ16-4で、未変換先頭ビット位置と検索中ビット位置が等しいと判断された場合は、ステップ16-9に分岐する。ステップ16-9では、次のステップ16-10で呼び出されるサブルーチンmake-same 処理で使用される変数c-bit の意味に合わせるために、c-bit にcon(p)を加算してさらに1減ずることにより、c-bit の内容を同様ビット後縁位置としている。ステップ16-10では、サブルーチンmake-same を呼び出して、図17で説明したように、様式①の変換データを格納し、ステップ16-11およびステップ16-12で、s-bitおよびc-bit を更新する。

【0036】ステップ16-3では、検索中ビット位置を示すポインタ変数c-bit が未変換先頭ビット位置変数s-bit に対して15多い（16ビット連続して連続同様ビットが存在しない）場合には、その時点の検索中行ビットまでのデータを変換するために、ステップ16-13に分岐し、サブルーチンmake-newを呼び出して、様式②に従ってデータを変換する。そしてステップ16-14およびステップ16-16で、s-bit およびc-bit を更新する。  
50

【0037】ステップ16-3で変数c-bitとs-bitの差が15未満の場合は、ステップ16-15に分岐し、検索中ビット位置を示すポインタ変数c-bitが最大ビット幅に達しているか否かを判断し、達している場合は処理を終了し、達していない場合はステップ16-16に分岐してc-bitを更新する。

【0038】図1におけるステップ1-8は、未変換ビットが最大ビット幅に達した場合にループを抜けるためのステップであり、未変換ビットが最大ビット幅に達した場合ステップ1-9に分岐し、それ以外の場合はステップ1-4に分岐する。ステップ1-9では、未変換ビットがある場合にそれらを様式②に変換するサブルーチンmake-newを呼び出し、処理を行う。

【0039】以上述べた変換アルゴリズムに基づいて図5に示す2進値表を変換した結果を図19に示す。本実施例により変換した場合は、960ビットのデータが696ビットに変換され、約7.2%に圧縮される。

【0040】

【発明の効果】以上詳細に説明したように本発明によれば、当該行のデータと先行する複数行のデータとを比較し、その中から当該行と同様のビットを最も長く有する行データを指定して圧縮するので、直前行のデータとほとんど一致しない印刷データもビット数が少なくなるように効率的に圧縮することができる。

【図面の簡単な説明】

【図1】本発明の実施例の動作を示すフローチャートである。

【図2】印刷結果を示す説明図である。

【図3】図2の印刷結果を得るための印刷データを示す説明図である。

【図4】他の印刷結果を示す説明図である。

【図5】図4の印刷データを示す説明図である。

【図6】圧縮データの様式①を示す説明図である。

【図7】圧縮データの様式②を示す説明図である。

【図8】第1行データを示す説明図である。

【図9】第2行データを示す説明図である。

【図10】第3行データを示す説明図である。

【図11】実施例を適用するプリンタのブロック図である。

【図12】実施例のサブルーチンを示すフローチャートである。

【図13】実施例のサブルーチンを示すフローチャートである。

【図14】実施例のサブルーチンを示すフローチャートである。

【図15】実施例のサブルーチンを示すフローチャートである。

【図16】実施例のサブルーチンを示すフローチャートである。

【図17】実施例のサブルーチンを示すフローチャートである。

【図18】実施例のサブルーチンを示すフローチャートである。

【図19】実施例により変換した結果を示す説明図である。

【符号の説明】

111 CPU

114 RAM

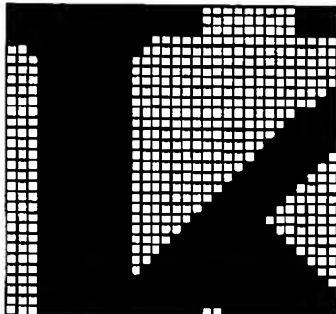
601 様式①

602 様式識別ビット

603 連続同様ビット数

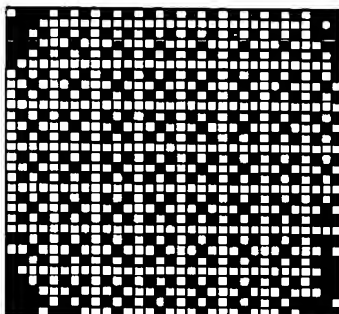
604 先行行数

【図2】



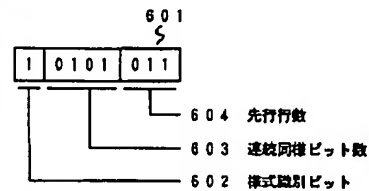
印刷結果を示す説明図

【図4】



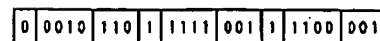
他の印刷結果を示す説明図

【図6】



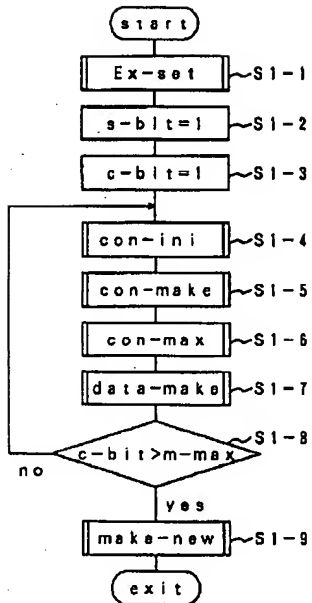
圧縮データの様式①を示す説明図

【図10】



第3行データを示す説明図

【図1】



実施例の動作を示すフローチャート

【図3】

第1行データ	11111111	11111111	11111000	00001111
第2行データ	11111111	11111111	11111000	00001111
	11111111	11111111	11111000	00001111
	11111111	11111100	00000000	00000000
	00111111	11111000	00000000	00000000
	00011111	11110000	00000000	00000000
	00011111	11110000	00000000	00000000
	00011111	11110000	00000000	00000000
	00011111	11110000	00000000	00000000
	00011111	11110000	00000000	00000001
	00011111	11110000	00000000	00000001
	00011111	11110000	00000000	00000011
	00011111	11110000	00000000	00000111
	00011111	11110000	00000000	00011111
	00011111	11110000	00000000	00111111
	00011111	11110000	00000000	01111111
	00011111	11110000	00000000	11111110
	00011111	11110000	00000001	11111100
	00011111	11110000	00000011	11111000
	00011111	11110000	00000111	11111000
	00011111	11110000	00001111	11110000
	00011111	11110000	00001111	11100000
	00011111	11110000	00011111	11000000
	00011111	11110000	00111111	10000000
	00011111	11110000	01111111	11000000
	00011111	11110000	11111111	11100000
	00011111	11110001	11111111	11110000
	00011111	11110011	11111111	11111000
	00011111	11110111	11111111	11111100
	00011111	11111111	11111111	11111110
	00011111	11111111	11111111	11111111
第30行データ	00011111	11111111	11111111	11111111
第1列データ	00011111	11111111	11100111	11111111

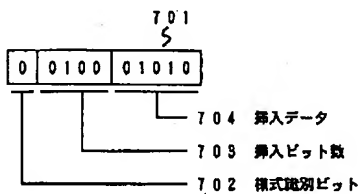
図2の印刷結果を得るための印刷データを示す説明図

【図5】

第1行データ	01110101	01010101	01010101	01010111
第2行データ	11100000	00000000	00000000	00000101
	11010101	01010101	01010101	01010111
	11100010	00100010	00100010	00100011
	11010101	01010101	01010101	01010101
	10000000	00000000	00000000	00000001
	01010101	01010101	01010101	01010101
	00100010	00100010	00100010	00100010
	01010101	01010101	01010101	01010101
	00000000	00000000	00000000	00000000
	01010101	01010101	01010101	01010101
	00100010	00100010	00100010	00100010
	01010101	01010101	01010101	01010101
	00000000	00000000	00000000	00000000
	01010101	01010101	01010101	01010101
	00100010	00100010	00100010	00100010
	01010101	01010101	01010101	01010101
	00000000	00000000	00000000	00000000
	01010101	01010101	01010101	01010101
	00100010	00100010	00100010	00100010
	01010101	01010101	01010101	01010111
	00000000	00000000	00000000	00000000
	11010101	01010101	01010101	01010111
	00100010	00100010	00100010	00100010
	11010101	01010101	01010101	01010111
	10000000	00000000	00000000	00000011
	11010101	01010101	01010101	01010111
	11100010	00100010	00100010	00100011
	11110101	01010101	01010101	01011110
第30行データ	11101110	00100010	00100010	00101111
第1列データ	01110101	11100010	00100010	00101111

図4の印刷データを示す説明図

【図7】



圧縮データの形式②を示す説明図

【図8】

0	1111	01110101	01010101	0	1111	01010101	01010111
---	------	----------	----------	---	------	----------	----------

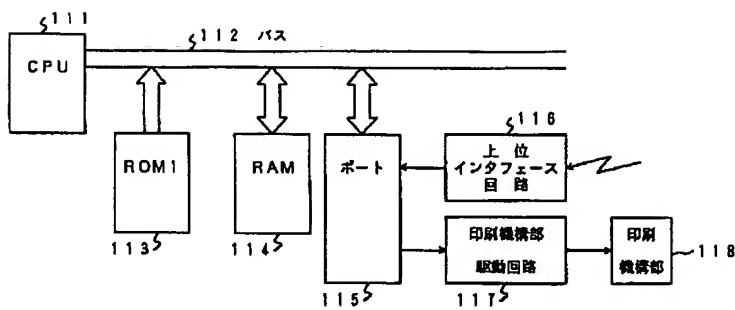
第1行データを示す説明図

【図9】

0	1111	11100000	00000000	0	1111	00000000	00000101
---	------	----------	----------	---	------	----------	----------

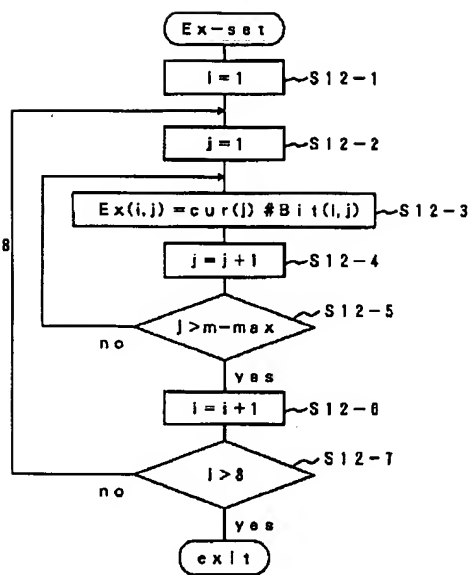
第2行データを示す説明図

【図11】



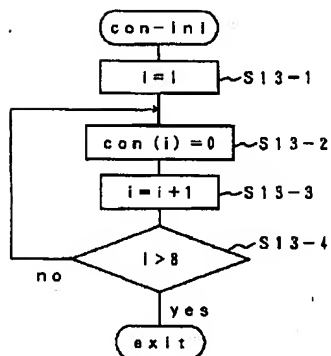
実施例を適用するプリンタのブロック図

【図12】



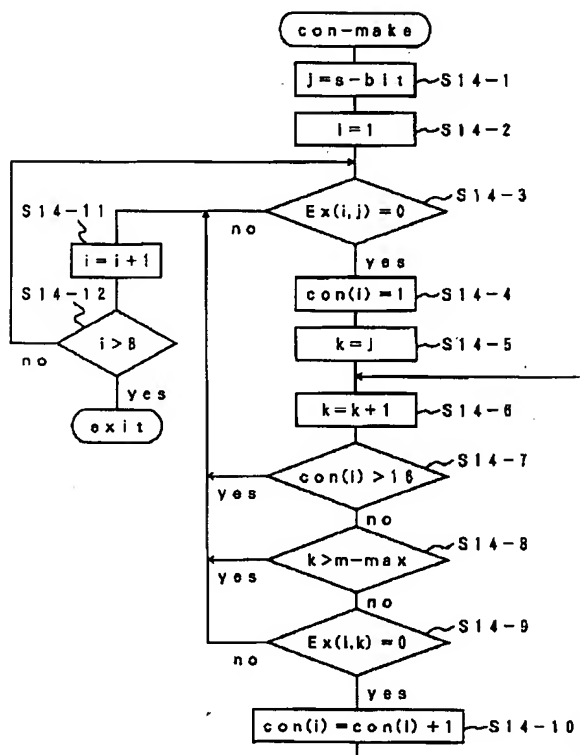
実施例のサブルーチンを示すフローチャート

【図13】



実施例のサブルーチンを示すフローチャート

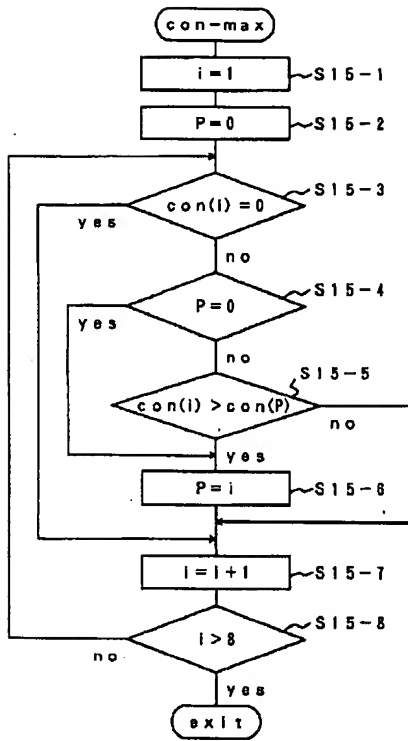
【図14】



実施例のサブルーチンを示すフローチャート

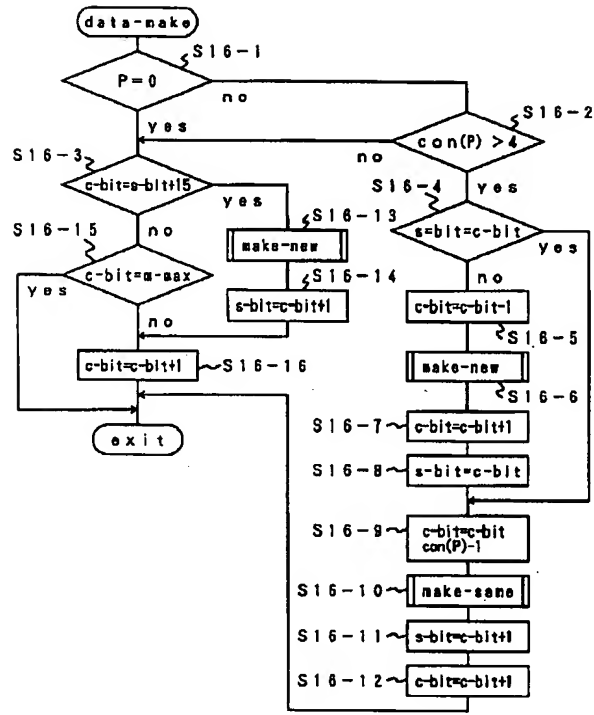


【図15】



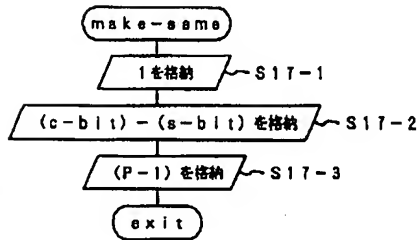
実施例のサブルーチンを示すフローチャート

【図16】



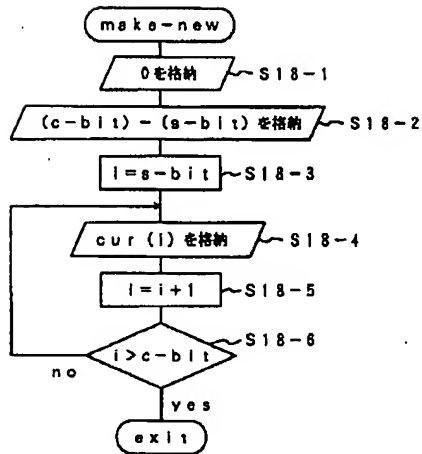
実施例のサブルーチンを示すフローチャート

【図17】



実施例のサブルーチンを示すフローチャート

【図18】



実施例のサブルーチンを示すフローチャート

【図19】

第1行データ		必要ビット数
1	0 1111 0110101 01010101 0 1111 01010101 01010111	42
2	0 1111 11100000 00000000 0 1111 00000000 00000101	42
3	0 0010 110 1 1111 001 1 1100 001	24
4	1 0101 001 0 1111 10001000 10001000 0 1001 10001000 11	44
5	1 1111 001 1 1011 001 0 0001 01	23
6	0 0010 100 1 1111 011 1 1001 011 0 0010 001	32
7	0 0000 0 1 1111 001 1 1110 001	22
8	0 0001 00 1 1111 011 1 1100 011 0 0000 0	29
9	1 1111 001 1 1111 001	16
10	0 0000 0 1 1111 0111 1011 011 0 0000 0	28
11	1 1111 001 1 1111 001	16
12	1 1111 011 1 1111 011	16
13	1 1111 001 1 1111 001	16
14	1 1111 011 1 1111 011	16
15	1 1111 001 1 1111 001	16
16	1 1111 011 1 1111 011	16
17	1 1111 001 1 1111 001	16
18	1 1111 011 1 1111 011	16
19	1 1111 001 1 1111 001	16
20	1 1111 011 1 1111 011	16
21	1 1111 001 1 1011 001 0 0001 11	23
22	1 1111 011 1 1111 011	16
23	0 0000 0 1 1111 001 1 1110 001	22
24	1 1111 011 1 1111 011	16
25	1 1111 001 1 1111 001	16
26	0 0000 1 1 1111 011 1 1100 011 0 0001 11	29
27	1 1111 001 1 1111 001	16
28	0 0001 11 1 1111 001 1 1100 011 0 0000 1	29
29	0 0010 111 1 1111 001 1 1000 001 0 0011 1110	33
30	0 0110 1110111 1 1111 001 0 1000 00010111 1	34

実施例により変換した結果を示す説明図